Randomized Algorithms: Methods and Techniques

Kuldeep Sharma
Assistant Professor
Computer Science & Engineering Department,
Chitkara University, H.P.

Dr. Deepak Garg
Senior Member IEEE
Computer Science &Engineering Department,
Thapar University, Patiala

ABSTRACT
Randomized Algorithms are now gaining the attention of researchers. The reason is that some of the randomized algorithms have been successfully implemented in important applications reducing the time complexity and other computing resources. This paper reviews the different methods and techniques available in randomized algorithms. Paper also gives the gaps in the existing research and the future scope of research in this area.

Keywords: Randomized Algorithms, LP Rounding, Monte Carlo.

1. INTRODUCTION
Everybody in the theory of computing community is well acquainted with the concept of randomization. It is not an exaggeration to say that randomization is currently one of the major approaches to algorithm design.

A randomized algorithm is an algorithm which typically uses the random input in the hope of achieving good performance in the "average case". Formally, the algorithm's performance will be a random variable determined by the random inputs, with good expected value. The "worst case" is typically so unlikely to occur that it can be ignored. Consider the problem of finding a P in an array of n elements, given that half are P's and the other half are Q's. The obvious approach is to look at each element of the array, but this would take very long (*n/2 operations*) if the array were ordered as P first followed by Q. There is a similar drawback with checking in the reverse order, or checking every second element. In fact, with any strategy at all in which the order in which the elements will be checked is fixed, i.e. a deterministic algorithm, we cannot guarantee that the algorithm will complete quickly for all possible inputs. On the other hand, if we were to check array elements at random, then we will quickly find P with high probability, whatever is the input. In the example above, the randomized algorithm always outputs the correct answer, it is just that there is a small probability of taking long to execute. Sometimes we want an algorithm which always complete quickly, but allow a small probability of error. Monte Carlo methods are methods of approximation of the solution to problems of computational mathematics, by using random processes for each such problem, with the parameters of the process equal to the solution of the problem. The method can guarantee that the error of Monte Carlo approximation is smaller than a given value with a certain probability. So, Monte Carlo methods always produce an approximation of the solution, but one can control the accuracy of this solution in terms of the probability error. The Las Vegas method is a randomized method which also uses random variable or random processes, but it always produces the correct result (not an approximation). The only variant is that it's running time might change between executions. A typical example is the well-known Quick sort method. Usually Monte Carlo methods reduce problems to the approximate calculation of mathematical expectations. Observe that any Las Vegas Algorithm can be converted into a Monte Carlo Algorithm, by having it output an arbitrary, possibly incorrect answer if it fails to complete within a specified time [1].

A randomized algorithm is one that receives, in addition to its input data, a stream of random bits that it can use for the purpose of making random choices. Even for a predetermined input, diverse runs of a randomized algorithm may give altered results; as a consequence it is inevitable that a description of the properties of a randomized algorithm will engage probabilistic statements. For example, even when the input is preset, the execution time of a randomized algorithm is a random variable. Behavior of randomized algorithm varies from one execution to another even with a fixed input.

Random variable is a function. For case in point, we can talk formally about these dice is to define the random variable Y1 representing the result of the first die, Y2 representing the result of the second die, and Y = Y1 + Y2 representing the sum of the two. We could then ask: what is the probability that Y = 7?

One property of a random variable we often care about is its expectation.

Randomized algorithms are tool in computational number theory; have by now found widespread application. Growth has been fuelled by the two major benefits of randomization one is its simplicity another one is speed. Numerous applications found that randomized algorithm is the fastest algorithm available, or the simplest in most of cases it is both. In the analysis of a randomized algorithm which establish bounds on the expected value of a performance measure (e.g., the running time of the algorithm) that are valid for every input; the distribution of the performance measure is on the random choices made by the algorithm based on the random bits provided to it [2].

1.1 Complexity Analysis
If one would like to consider Exp-Time$_A$ as a function of the input size, then one uses the worst case approach, i.e., the expected time complexity of A is

Exp-Time$_A$(n)= max { Exp-Time$_A$(x) | x is an input of size n} for every n $\in N$

It is often not easy to analyze Exp-Time$_A$(n) for a given randomized algorithm A. To overcome this difficulty one also uses the worst case approach from the beginning. This means:

Time$_A$(x) = max {Time(C) | C is a run of A on x}.

Then, the (worst case) time complexity of A is

$\boldsymbol{Time_A(n)}$= max { TimeA(x) | x is an input of size n }.

This definition may be misleading in some cases. This is because randomized algorithms may allow infinite runs provided that they occur with a reasonably small probability on any given input.

Randomized algorithms are better than deterministic one's simple example is Quick Sort.

Randomized Quick sort (RQS)

Input: $S = \{a_1. \ldots a_n\}$

Step 1: *Choose an* $i \in \{1 \ldots n\}$ *uniformly at random*

Step 2: *if n = 1 output(S)*

$S<:= \{b \in S | b < a_i\}$

else

$S=:= \{b \in S | b = a_i\}$

$S>:= \{b \in S | b > a_i\}$

Step 3: *Recursively sort* $S_<$ *and* $S_>$

Output: $RQS(S_<), S=, RQS(S_>)$

In simple algorithm the worst case complexity is $O(n^2)$ where in Randomized Algorithm its
$O(n \log n)$ [3].

2. DESIGN PARADIGMS OF RANDOMIZED ALGORITHMS

2.1 Abundance of Witnesses

Time and again a computational problem required a witness or a certificate that capably authenticate a hypothesis. Say, a number x has certain property i.e. x is prime; to justify this property it need proof (witness) but for many problems, the witness lies in a search space that is too large to be searched exhaustively. However, if the search space were to contain a relatively large number of witnesses, a randomly chosen element is likely to be a witness. Further, autonomous repetitions of the sampling reduce the probability that a witness is not found on any of the repetitions. The most prominent examples of this phenomenon occur in number theory. Indeed, the problem of testing a given integer for primality has no known deterministic polynomial-time algorithm. There are, however, several randomized polynomial-time algorithms [4] that will, on any input, correctly perform this test with high probability.

2.2 Fingerprinting and Hashing

A fingerprint is the image of an element mapping into another Fingerprints obtained via random mappings have many useful properties. For example, in pattern-matching applications [5] it can be shown that two strings are likely to be identical if their fingerprints are identical; comparing the short fingerprints is considerably faster than comparing the strings themselves. Another example is hashing [6], where the elements of a set S are stored in a table of size linear in uSu with the guarantee that the expected number of elements in S mapped to a given location in the table is $O(1)$. This leads to efficient schemes for deciding membership in S. Random fingerprints have found a variety of applications in generating pseudorandom numbers and complexity theory (for instance, the verification of algebraic identities [7]).

2.3 Foiling an Adversary

In the classical worst-case analysis of deterministic algorithms, a lower bound is established on the running time of algorithms by postulating an "adversary" that constructs an input on which the algorithm fares poorly. The input thus constructed may be different for each deterministic algorithm. With a game-theoretic interpretation of the relationship between an algorithm and an adversary, we can view a randomized algorithm as a probability distribution on a set of deterministic algorithms. (This observation underlies Yao`s [8] adaptation of von Neumann`s Mini-Max Theorem in game theory into a technique for establishing limits on the performance improvements possible via the use of a randomized algorithm.) Although the adversary may be able to construct an input that foils one (or a small fraction) of the deterministic algorithms in the set, it may be impossible to devise a single input that is likely to defeat a randomly chosen algorithm. For example, consider a uniform binary AND-OR tree with n leaves. Any deterministic algorithm that evaluates such a tree can be forced to read the Boolean values at every one of the n leaves. However, there is a simple randomized algorithm [9] for which the expected number of leaves read on any input is $O(n^{0.794})$.

2.4 Random Sampling

A pervasive theme in randomized algorithms is the idea that a small random sample from a population is representative of the population as a whole. Because computations involving small samples are inexpensive, their properties can be used to guide the computations of an algorithm attempting to determine some feature of the entire population. For instance, a simple randomized algorithm [10] based on sampling finds the k^{th} largest of n elements in $1.5n + O(n)$ comparison steps, with high probability. In contrast, it is known that any deterministic algorithm must make at least $2n$ comparisons in the worst case.

2.5 Rapidly Mixing Markov Chains

Markov chains are probability models for trials of random experiments of great variety, and their defining characteristic is that they allow us to consider situations where the future evolution of the process of interest depends on where it is at present, but not on how it got there. This contrasts with the independent trials models we have measured in the law of large numbers and the central limit theorem. For independent trial processes the possible outcomes of each trial of the experiment are the same and occur with the same probability. Furthermore, what happens on any trial is not affected by what happens on any other trial. With Markov chain models we can generalize this to the extent that we allow the future to depend on the present [11].

In counting problems, the goal is to determine the number of combinatorial objects with a specified property. When the space of objects is large, an appealing solution is the use of the Monte Carlo approach of determining the number of desired objects in a random sample of the entire space. In a number of cases, it can be shown that picking a uniform random sample is as difficult as the counting problem itself. A particularly successful technique for dealing with such problems is to generate near uniform random samples by defining a Markov chain on the elements of the population, and showing that a short random walk using this Markov chain is likely to sample the population uniformly.

This method is at the core of a number of algorithms used in statistical physics. Examples include algorithms for estimating the number of perfect matching's in a graph [12, 13].

3. RANDOMIZED DEPENDENT LP-ROUNDING ALGORITHM

The application and new properties of dependent rounding technique in the domain of facility location which uses methods for uncapacitated facility location. LP-rounding approximation algorithms for facility location problems are based on partitioning facilities into disjoint clusters and opening at least one facility in each cluster. They extend this approach which provides a quite tight analysis resulting in the improved approximation ratio. They construct a laminar family of clusters, which then guides the rounding procedure. It allows exploiting properties of dependent rounding, and provides a quite tight analysis resulting in the improved approximation ratio [14]. They gave a new randomized LP-rounding 1.725- approximation algorithm for the metric Fault-Tolerant uncapacitated Facility Location Problem. This improves on the previously best known 2.076 approximation algorithm of Swamy & Shmoys.

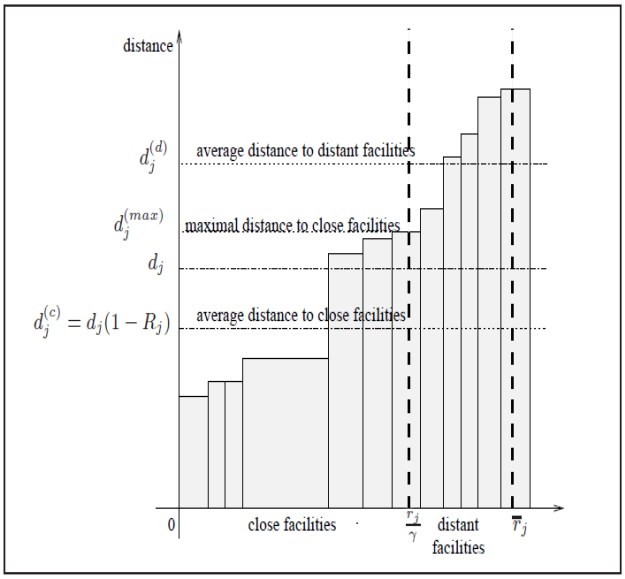

Fig 1. Distances to facilities serving clients

4. RANDOMIZED: ALL PAIR SHORTEST PATH

Let G (V,E) be an undirected, connected graph with V={1…..n} and |E|=m. The adjacency matrix *A* is *n* x *n 0-1* matrix with $A_{ij} = A_{ji} = 1$ if and only if the edge *(i,j)* is present in E. Given A, we define the distance matrix D and n x n matrix with non-negative integer entries such that D_{ij} equals the length of a shortest path from vertex *i* to vertex *j*. The diagonal entries in both A and D are zeroes. Since G is connected, all entries in D are finite; this is not a restrictive assumption since a graph can be decomposed easily into connected components in linear time. The all-pair shortest path problem is to compute a representation of the shortest paths between all pairs of vertices, i.e., the paths that determine the entries in the distance matrix.

The all-pair shortest paths problem can be solved in *O (nm)* time, as follows: from each vertex $i \in V$, compute the breadth-first search tree T_i rooted at *i.* Each such tree can be computed in *O (m)* time and in any tree T_i the (unique) path from *i to* any vertex j is the shortest path between them. Given the collection of breadth-first search trees the distance matrix can be computed in *O* (n^2) time by assigning level numbers to the vertices in each tree. The class of algorithms of Dijkstra, Floyd- Warshall and Johnson solve all-pair shortest paths in *O* (n^3), where as in randomized its *O (MM (n) log*^{2}*n).* [15].

5. CONCLUSION AND FUTURE SCOPE

There are many problems whose deterministic algorithm is available but still there is scope to find the randomized algorithm. In case of many randomized algorithm available the lower bound has been proved but algorithm of that complexity are not available still there is scope to improve the gap. There are some problems whose Monte Carlo algorithm is available the Las Vegas algorithm is not available and vice-versa. There is still scope to formulate an algorithm for the all-pairs shortest paths problem that does not use matrix multiplication and runs in time *O* ($n^{3-\varepsilon}$) for a positive constant ε. Devising an algorithm for computing the diameter of an un-weighted graph that does not use matrix multiplication and runs in time $O(n^{3-\varepsilon})$ for a positive constant ε. Work can be done in devising a simple randomized MST verification algorithm with expected running time *O(n+m).*

6. REFERENCES

[1] Donald Knuth.1997 The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley,. ISBN 0-201-89685-0. Pages 113–122 of section 5.2.2: Sorting by Exchanging

[2] Rajeev Motwani and Prabhakar Raghavan: 1996 Randomized Algorithms in ACM Computing Surveys Vol.28, No.1 Page no 33-37.

[3] Hoare, C. A. R.1961 Partition: Algorithm 63, Quicksort: Algorithm 64, and Find: Algorithm 65. Comm. ACM 4(7), 321-322,

[4] Solovay, R. and Strassen, V. 1977. A fast Monte-Carlo test for primality. SIAM J. Comput. 6, 84–85 & SIAM J. Comput. 7, 1 (Feb.), 1978, 118.

[5] Karp R.M., Rabin M.O., 1987, Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. 31(2):249-260.

[6] Carter, Larry; Wegman, Mark N. 1979. Universal Classes of Hash Functions. Journal of Computer and System Sciences 18 (2): 143–154.

[7] Freivalds, R. 1977. Probabilistic machines can use less running time. In Information Processing 77, Proceedings of IFIP Congress 77, Gilchrist, Ed., (Aug.), North-Holland, Amsterdam, 839–842.

[8] Yao,A.C-C.1977.Probabilistic computations Towards a unified measure of complexity. In Proceedings of the 17th Annual Symposium on Foundations of Computer Science, 222–227.

[9] R.W and Rivest 1975. Expected time bounds for selection. Commun. ACM 18, 165–172.

[10] Snir,M. 1985. Lower bounds on probabilistic linear decision trees. Theory of Computer Science. 38, 69–82.

[11] Sinclair, A. 1992. Algorithms for Random Generation and Counting: A Markov Chain Approach. Progress in Theoretical Computer Science. Birkhauser, Boston.

[12] Wai Ki Ching, Michael K. Ng- 2006 Markov Chains: Models, algorithm and applications ISBN-10:0-387-29335-3

[13] Karp, R.M :1991 AN introduction to randomized algorithm, Discrete Applied Mathematics 165-201.

[14] Jaroslaw Byrka, Aravind Srinivasan, Chaitanya Swamy: 2010 Fault-Tolerant Facility Location: a randomized dependent LP-rounding algorithm CoRR abs/ 1003.1295

[15] Floyd, R.W and Rivest, R.L:1975. Expected time bounds for selection. Commun. ACM 18, 165–172.